An Open-Source Hardware/Software Architecture for Quadrotor UAVs

Riccardo Spica, Paolo Robuffo Giordano, Markus Ryll, Heinrich H Bülthoff, Antonio Franchi

An Open-Source Hardware/Software Architecture for Quadrotor UAVs

Riccardo Spica * **Paolo Robuffo Giordano** ** **Markus Ryll** ***
Heinrich H. Bülthoff *** **Antonio Franchi** ***

* *Inria Rennes Bretagne Atlantique and IRISA, Campus de Beaulieu, 35042 Rennes Cedex, France; (e-mail: riccardo.spica@irisa.fr).*
** *CNRS at IRISA, Campus de Beaulieu, 35042 Rennes Cedex, France (e-mail: prg@irisa.fr)*
*** *Max Planck Institute for Biological Cybernetics, Spemannstraße 38, 72076 Tübingen, Germany, (e-mail: antonio.franchi@tuebingen.mpg.de)*

Abstract: In this paper, we illustrate an open-source ready-to-use hardware/software architecture for a quadrotor UAV. The presented platform is price effective, highly customizable, and easily exploitable by other researchers involved in high-level UAV control tasks and for educational purposes as well. The use of object-oriented programming and full support of Robot Operating System (ROS) and Matlab Simulink allows for an efficient customization, code reuse, functionality expansion and rapid prototyping of new algorithms. We provide an extensive illustration of the various UAV components and a thorough description of the main basic algorithms and calibration procedures. Finally, we present some experimental case studies aimed at showing the effectiveness of the proposed architecture.

1. INTRODUCTION

Quadrotors are Unmanned Aerial Vehicles (UAVs) actuated by four fixed-pitch independent propellers located at the vertexes of a cross-shaped structure [11]. In addition to being able to take-off and land vertically, quadrotors can reach high angular accelerations thanks to the relatively long lever arm between opposing motors. This makes them more agile than most standard helicopters or similar rotorcraft UAVs. For these reasons, and also because of their affordability and mechanical robustness, this platform has witnessed a considerable growing interest in the robotics and hobbyist community over the last decade, being, in some cases, also commercialized as off-the-shelf products [1, asc].

Nowadays, most of the research efforts are not dealing with more accurate dynamical modeling, or novel flight control design, as these issues have reached an adequate level of maturity and do not need further major improvements, at least concerning standard applications. The challenge is rather on how to exploit these systems as a flexible platform for implementing and validating complex and higher-level tasks, often involving multiple robots interacting at the same time in partially structured or unstructured environments. Some examples in this sense can be found in, e.g., [14, 12, 2].

Nevertheless, obtaining a reliable and operative quadrotor platform for experimental purposes still requires a considerable investment of time and resources for properly tuning the system, e.g., for taking care of identification and calibration of all the various parameters (offsets, biases, motor curves), gain tuning of the flight controller, implementation of accurate onboard state estimation filters, and, last but not least, software development and debugging. These issues are obviously common to any robot, but, we believe, are particularly sensitive for flying robots as the malfunctioning of any hardware/software component can easily result in a severe crash and loss of the vehicle. Because of these reasons, many ready-to-use platforms have been proposed over the last years, each of them with their pros/cons and spanning a wide range of prices.

The Hummingbird and Pelican quadrotors by Ascending Technologies are probably the most widespread platforms in the research community (see, e.g., [14, 12]). They are characterized by a solid flight performance thanks to their lightweight rigid structure, but their price is still expensive compared to other solutions. An alternative consumer-oriented and cheaper quadrotor platform, employed in some research projects, is the AR.Drone Parrot [1] which is able to achieve a stable near hovering flight only using its onboard sensors. However, being the AR.Drone an inexpensive quadrotor designed for the general consumer market, it possesses clearly some drawbacks when used within a research project, e.g., 1) it does not allow for the tuning and customization that are often required in research projects, 2) it is not possible to integrate additional sensors and high performance computational units (due to the extremely low payload and fragile structure), and 3) the usability of the integrated sensors for testing new algorithms is limited by their quality and the latencies at which the corresponding signals can be accessed. Furthermore, both these architectures are closed-source and the low-level controller cannot be freely customized to comply with the users' needs.

As for open-source architectures, an interesting survey of some existing projects can be found in [7]. Many of the referenced solutions are extremely competitive when

compared to their price. Nevertheless, in our opinion none of them possesses enough computational resources to run onboard the typical complex algorithms required for a truly autonomous operation of quadrotor UAVs, such as, e.g., intensive image processing and/or high load communication management. Another project, similar to the one presented in this paper, is described in [13]. The authors exploit the same hardware used in this work, but they do not customize the low-level controller provided by the manufacturer and strongly rely on a ground-based computer for all computations.

With respect to all these solutions, the goal of this paper is to propose a open-source and relatively cheap architecture for a quadrotor UAV characterized by the basic hardware and software components that can be used within an educational and research project to perform autonomous flight and onboard implementation of complex behaviors. For this purpose we are making the complete source code and specs available on a freely-accessible repository for dissemination purposes and easy re-use by other research groups (see `https://svn.kyb.mpg.de/kyb-robotics/`). From a hardware point of view, the proposed platform is purposely simple, highly customizable and easy to maintain. Every component can be bought off-the-shelf, and the use of standard software architectures and protocol allows for some flexibility in the hardware setup. As from a software point-of-view, the use of object-oriented programming and the support of the Robot Operating System (ROS) and Matlab Simulink environments allow for efficient customization, code reuse, functionality expansion and rapid prototyping of new algorithms. Finally, for the sake of completeness and re-usability, we also give hands-on details on all the calibration procedures necessary for identifying the dynamic parameters of the quadrotor, and on its basic flight control algorithms.

The paper is organized as follows: Secs. 2–3 describe the general hardware and software architecture of the proposed quadrotor platform. Then, Secs. 4–6 provide details on the implemented trajectory tracking controller, relevant calibration procedures, and state estimation schemes. Finally, Sec. 7 reports some experimental results for some relevant test cases meant to illustrate the potential and performance of the proposed platform, and Sec. 8 draws some conclusions and discusses future directions.

2. HARDWARE DESIGN

The main goals driving our hardware design can be summarized as follows:

- keeping the cost of the overall components as low as possible, when compared to equivalent solutions,
- guaranteeing a wide availability of all the components by avoiding the use of customized hardware,
- ensuring an adequate onboard computational power, for the reduced payload of a typical small-size quadrotor platform,
- ensuring that both the actuation and sensing capabilities of the platform are made fully interfaced with the ROS environment.

Taking into account the aforementioned specifications, we found a suitable solution in using the mechanical frame, actuators, microcontrollers, and inertial measurement unit (IMU) of the *MK-Quadro* [1] platform, by nevertheless replacing the control layer provided by the manufacturer, introducing new components, and completely redesigning the interface to the platform actuation and sensing resources.

The actuation system of the MK-Quadro consists of four plastic propellers with a diameter of 0.254 m, and a total span and weight of the frame of 0.5 m and 0.12 kg, respectively. The avionics, attached to the center plate of the frame, consists of: *i*) a Pico-ITX LP-170C motherboard by COMMELL, equipped with a dual core Intel® Atom™ 64 bit 1.80 GHz CPU, 2 GB DDR2 RAM, 16 GB Compact Flash Type-II memory and a PCIE mini wireless board, *ii*) a *low-level* 8-bit Atmega1284p microcontroller, clocked at 20 MHz, connected to the mini-computer trough two RS232 serial ports and a MAX232 converter. The serial connections operate at a baud-rate of 115 200 Bd *iii*) four brushless controllers connected to the low-level controller through a standard I^2C bus, *iv*) three 3D LIS344alh accelerometers ($0.0039 g_0$ m/s^2 resolution and $\pm 2 g_0$ m/s^2 range) and three ADXRS610 gyros (0.586 deg/s resolution and ± 300 deg/s range), directly connected to the analog to digital 10 bit converters of the low-level microcontroller. In addition to this, the platform is also equipped with a pressure sensor. The whole system is powered by a 2600 mAh LiPo battery which guarantees an endurance of around 10 min of flight in normal regimes. The complete system has a weight of approximately 1.300 kg.

On top of this basic setup the system permits the easy addition of onboard exteroceptive sensor, as done, e.g., in [3, 2] by using onboard monocular cameras. The use of an onboard RGB-D sensor is currently under development.

We finally note that this hardware solution should be regarded as a *suggested configuration*, i.e., as a well tested and consolidated, rather than as a mandatory one. Indeed, other hardware choices are possible: in particular, the high-level control software can be run on any standard x86/64 architecture without additional effort. Moreover, since the source code is freely distributed, it is possible (with, probably, some more effort in this case) to compile and run it on different architectures. Finally, concerning the mechanical components, any equivalent configuration can be used by suitably adjusting the control/estimation gains.

3. SOFTWARE ARCHITECTURE

The software solution developed for our platform is meant to comply with the following specifications:

- possibility of distributing the full open-source code,
- use of an object-oriented high-level programming language (such as C++ or Python),
- possibility of providing the user with a working system in terms of flight control, making the platform ready-to-use in laboratory conditions so as to, e.g., test new state estimation and planning algorithms for exploration or multi-robot formation controllers,
- possibility to customize, if desired, the single basic functionalities, such as the controllers or the basic state estimator provided with the platfrom,

[1] http://www.mikrokopter.de

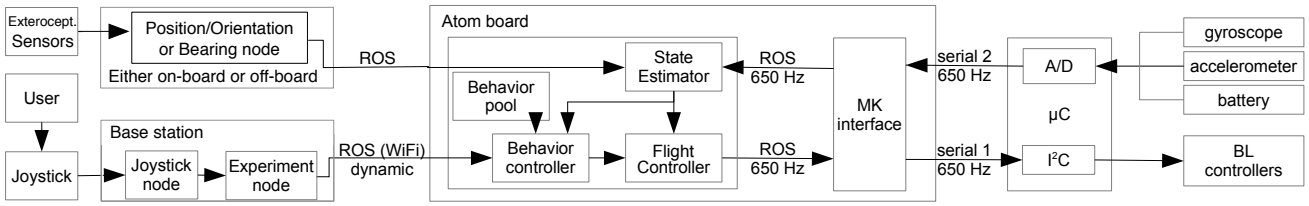


Fig. 1. Block diagram of the hardware and software architecture. The TeleKyb framework is used for the software part.

- possibility to automatize as much as possible the calibration procedure,
- easy integration of additional functionalities in the architecture, and easy interface to Matlab and Simulink.

In order to comply with the first two constraints, we opted for Ubuntu Linux Server 12.04 LTS as operating system, as this distribution is currently among the most widespread ones. Furthermore, for facilitating the integration and/or replacement of code parts, we adopted the ROS framework: indeed, this is more and more becoming a standard for robotic applications, with many common functionalities and drivers freely available on the web.

The overall system architecture is represented by the block diagram in Fig.1. The whole architecture is developed within the the *TeleKyb* framework (see [4] for more details). TeleKyb manages the generation of the desired trajectories from a pool of *behaviors* and selects the current state estimator and the motion controller. The control loop frequency depends on the kind of controller used and may vary, e.g., from 60 Hz to 650 Hz. Both the state estimation and the control algorithms are incapsulated in separate plugins. It is therefore easy, if desired, to re-implement them in a customized way.

Behaviors describe the basic operations available for the UAV (e.g., taking-off, landing, following a trajectory, executing velocity commands from a joystick, and so on) at a high level and in a parametrized way. Selection of the behaviors is governed by the *behavior controller* according to the robot state, to the experimental flow (managed by the *experiment node*), and to the joystick inputs from the user.

The *experiment node* is an independent ROS node running in a remote base station and communicating with the Atom board through a standard WiFi connection. From here, the user can control the experiment, modify the parameters online and log data. Finally the base station can also run Matlab Simulink models able to exchange data with the other software components via ROS publishers and subscribers instantiated in s-function blocks.

The *MkInterface node* is responsible for the communication with the low-level Atmega1284p microcontroller throughout the two serial ports. It receives the IMU measurements, the barometer measurements, the battery status, and the value of other internal variables. On the other hand it allows both to send control commands and change the logical states of the microcontroller. This latter can operate in two different modalities:

- In the *near-hovering* mode, the microcontroller receives, from the high-level control loop, a desired total thrust force, roll and pitch angles and yaw rate. It regulates these quantities using the information provided by the IMU and employing a standard linearized hovering controller that is described in [5];
- In the *direct* mode, instead, the microcontroller is only acting as a communication interface with the sensory and actuation hardware. It collects the IMU readings and then forwards the motor velocity commands (from the high-level control loop) to the brushless (BL) controllers via the I^2C bus.

Additionally, an *emergency* mode is also possible: transitions to this mode can be toggled by either the user or by the occurrence of dangerous situations such as, e.g., a very low battery level or an interruption of the communication link with the high-level control. In this case, the microcontroller ignores any further command from the Atom board and tries to land by only relying on the near-hovering controller.

Finally, in all cases the microcontroller runs an onboard state estimation based on a complementary filter as described in [9]. This estimation is not strictly necessary in the direct mode but, in order to obtain a smooth transition to the emergency mode at any time, the estimation is nevertheless always kept updated.

4. MODELING AND CONTROL

In this section we provide details on the implemented control algorithm for trajectory tracking when the microcontroller is operating in *direct* mode. As typically done, we neglect the dynamics of the propeller actuation and consider the generated thrusts as the actual control inputs for control design. As for the control scheme itself, we chose to employ a geometric tracking technique similar to the one presented in [6] because of its almost global convergence and excellent tracking capabilities.

Throughout the following, a right subscript will indicate the reference frame to which the quantities belong, and a left superscript the frame in which the quantities are expressed. An omitted left superscript will indicate a quantity expressed in the world inertial frame $\mathcal{W}$. The mapping between the thrusts f_i generated by each propeller along its spinning axis and the resulting total thrust u_1 along the vertical axis of the robot, and the three independent torques $u_{2,3,4}$ actuating the quadrotor body, is:

$$\mathbf{u} = \begin{pmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 0 & l & 0 & -l \\ -l & 0 & l & 0 \\ c & -c & c & -c \end{pmatrix} \begin{pmatrix} f_1 \\ f_2 \\ f_3 \\ f_4 \end{pmatrix} = \mathbf{M}\tilde{\mathbf{u}}, \qquad (1)$$

where l represents the distance of the propeller rotational axes from the center of the robot frame and c is a constant related to the propeller characteristics.

The robot state consists of the position and orientation of the body frame $\mathcal{B}$ w.r.t. the world inertial frame $\mathcal{W}$, namely $\boldsymbol{r}_{\mathcal{B}}$ and ${}^{\mathcal{W}}\boldsymbol{R}_{\mathcal{B}}$, together with the corresponding time-derivatives, i.e. the linear and angular velocity ($\dot{\boldsymbol{r}}_{\mathcal{B}}$ and ${}^{\mathcal{B}}\boldsymbol{\omega}_{\mathcal{BW}}$). Assuming that the robot barycenter is positioned in the geometric center, we obtain

$$m\ddot{\boldsymbol{r}}_{\mathcal{B}} + mg\boldsymbol{z}_{\mathcal{W}} = u_1\boldsymbol{z}_{\mathcal{B}}, \tag{2}$$

$$\boldsymbol{J}\,{}^{\mathcal{B}}\dot{\boldsymbol{\omega}}_{\mathcal{BW}} + [{}^{\mathcal{B}}\boldsymbol{\omega}_{\mathcal{BW}}]_{\times}\boldsymbol{J}\,{}^{\mathcal{B}}\boldsymbol{\omega}_{\mathcal{BW}} = (u_2, u_3, u_4)^T, \tag{3}$$

where m and $\boldsymbol{J}$ are the total mass and the (constant) inertia tensor of the robot respectively, $\boldsymbol{z}_{\mathcal{B}}$ is the Z-axis of the body frame and $[{}^{\mathcal{B}}\boldsymbol{\omega}_{\mathcal{BW}}]_{\times}$ is the skew-symmetric matrix associated to vector ${}^{\mathcal{B}}\boldsymbol{\omega}_{\mathcal{BW}}$.

Given a desired trajectory $(\boldsymbol{r}_{\mathcal{B},t}, \psi_t)$ for the robot position and yaw angle with all necessary derivatives, and having defined the position and velocity errors as $\boldsymbol{e}_p = \boldsymbol{r}_{\mathcal{B},t} - \boldsymbol{r}_{\mathcal{B}}$ and $\boldsymbol{e}_v = \dot{\boldsymbol{r}}_{\mathcal{B},t} - \dot{\boldsymbol{r}}_{\mathcal{B}}$, we can introduce a *desired force* as a PID action plus gravity cancellation

$$\boldsymbol{f}_d = m\ddot{\boldsymbol{r}}_{\mathcal{B},t} + \mathbf{K}_p^{hl}\boldsymbol{e}_p + \mathbf{K}_{\int p}^{hl}\int_0^t \boldsymbol{e}_p dt + \mathbf{K}_v^{hl}\boldsymbol{e}_v + mg\boldsymbol{z}_{\mathcal{W}}, \tag{4}$$

where $\mathbf{K}_p^{hl}$, $\mathbf{K}_{\int p}^{hl}$ and $\mathbf{K}_v^{hl}$ are positive diagonal matrices. The desired force $\boldsymbol{f}_d$ can be realized by choosing $u_1 = \boldsymbol{f}_d^T\boldsymbol{z}_{\mathcal{B}}$ and by aligning the body vertical axis with the direction of $\boldsymbol{f}_d$. The desired robot attitude, described by the rotation matrix ${}^{\mathcal{W}}\boldsymbol{R}_{\mathcal{B},d}$, can then be chosen so that its third column is $\boldsymbol{z}_{\mathcal{B},d} = \frac{\boldsymbol{f}_d}{\|\boldsymbol{f}_d\|}$. The remaining degrees of freedom are then set so as to minimize the yaw error. This is obtained by defining

$$\boldsymbol{y}_{\mathcal{C},d} = \left(-\sin\psi_t \;\; \cos\psi_t \;\; 0\right)^T,$$

and by choosing the first and second columns of ${}^{\mathcal{W}}\boldsymbol{R}_{\mathcal{B},d}$ as:

$$\boldsymbol{x}_{\mathcal{B},d} = \frac{\boldsymbol{y}_{\mathcal{C},d} \times \boldsymbol{z}_{\mathcal{B},d}}{\|\boldsymbol{y}_{\mathcal{C},d} \times \boldsymbol{z}_{\mathcal{B},d}\|}, \quad \boldsymbol{y}_{\mathcal{B},d} = \boldsymbol{z}_{\mathcal{B},d} \times \boldsymbol{x}_{\mathcal{B},d}.$$

The orientation error can then be defined as

$$\boldsymbol{e}_R = \frac{1}{2}\left({}^{\mathcal{W}}\boldsymbol{R}_{\mathcal{B}}^T\,{}^{\mathcal{W}}\boldsymbol{R}_{\mathcal{B},d} - {}^{\mathcal{W}}\boldsymbol{R}_{\mathcal{B},d}^T\,{}^{\mathcal{W}}\boldsymbol{R}_{\mathcal{B}}\right)^{\vee},$$

where $\vee$ indicates the *vee-map* that relates a skew-symmetric matrix to its corresponding 3-vector. The angular velocity error is instead

$$\boldsymbol{e}_{\omega} = {}^{\mathcal{W}}\boldsymbol{R}_{\mathcal{B}}^T\,{}^{\mathcal{W}}\boldsymbol{R}_{\mathcal{B},d}\,{}^{\mathcal{B},d}\boldsymbol{\omega}_{\mathcal{BW},d} - {}^{\mathcal{B}}\boldsymbol{\omega}_{\mathcal{BW}}$$

where ${}^{\mathcal{B},d}\boldsymbol{\omega}_{\mathcal{BW},d}$ can be computed as a function of the specified trajectory of the robot barycenter and its derivatives up to the third order (see [16]). Finally the torque inputs can be chosen as

$$\begin{aligned}(u_2, u_3, u_4)^T = {} & \mathbf{K}_R^{hl}\boldsymbol{e}_R + \mathbf{K}_{\int R}^{hl}\int_0^t \boldsymbol{e}_R dt + \mathbf{K}_{\omega}^{hl}\boldsymbol{e}_{\omega} + [{}^{\mathcal{B}}\boldsymbol{\omega}_{\mathcal{BW}}]_{\times}\boldsymbol{J}\,{}^{\mathcal{B}}\boldsymbol{\omega}_{\mathcal{BW}} \\ & + \boldsymbol{J}\left({}^{\mathcal{W}}\boldsymbol{R}_{\mathcal{B}}^T\,{}^{\mathcal{W}}\boldsymbol{R}_{\mathcal{B},d}\,{}^{\mathcal{B},d}\dot{\boldsymbol{\omega}}_{\mathcal{BW},d} - [{}^{\mathcal{B}}\boldsymbol{\omega}_{\mathcal{BW}}]_{\times}\,{}^{\mathcal{W}}\boldsymbol{R}_{\mathcal{B}}^T\,{}^{\mathcal{W}}\boldsymbol{R}_{\mathcal{B},d}\,{}^{\mathcal{B},d}\boldsymbol{\omega}_{\mathcal{BW},d}\right)\end{aligned} \tag{5}$$

where $\mathbf{K}_R^{hl}$, $\mathbf{K}_{\int R}^{hl}$ and $\mathbf{K}_{\omega}^{hl}$ are positive diagonal matrices. Note that this control law essentially consists of a PID action, a cancellation of the gyroscopic force and a feed-forward term ${}^{\mathcal{B},d}\dot{\boldsymbol{\omega}}_{\mathcal{BW},d}$ that can be computed as a function of the specified trajectory of the robot barycenter and its derivatives up to the fourth order (see again [16]).

At this point, since matrix $\mathbf{M}$ in (1) has full rank, it can be inverted to obtain the single motor thrusts f_i. As well-known, and shown in Sec. 5, these are related to the spinning velocities of the propellers which are then treated as references for the motor controllers. Nevertheless, directly feeding these references may conflict with any internal saturations due to the limited motor power. The resulting actuation can therefore significantly differ from the desired one. It is then necessary to properly scale down the control inputs in order to make them feasible by the motor controllers. We can achieve this goal by splitting the reference control input $\mathbf{u}$ in a component $\mathbf{u}_g = (mg, 0, 0, 0)^T$, necessary for compensating gravity in hovering, and a second component $\mathbf{u}_{res} = \mathbf{u} - \mathbf{u}_g$ on which a uniform scaling is applied until all the resulting propeller thrusts lie in the admissible range $[f_{min}, f_{max}]$. In practice one has to solve the following optimization problem:

$$\begin{aligned}\min_{\{\lambda\}} \quad & \frac{1}{2}\left\|\mathbf{u} - (\mathbf{u}_g + \lambda\mathbf{u}_{res})\right\|^2 \\ \text{s.t.} \quad & \mathbf{f}_{min} \leq \mathbf{A}^{-1}(\mathbf{u}_g + \lambda\mathbf{u}_{res}) \leq \mathbf{f}_{max}, \quad \lambda \leq 1.\end{aligned}$$

Note that an admissible solution always exists, provided that $\mathbf{u}_g$ is a feasible input. Moreover the use of a uniform scaling guarantees that the direction of the control input is maintained.

5. CALIBRATION AND IDENTIFICATION

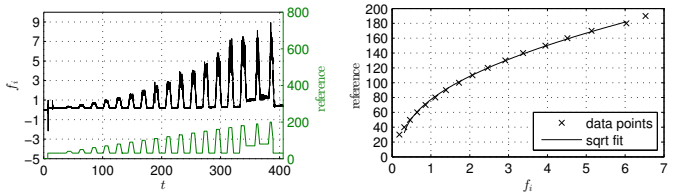
A necessary condition for obtaining good flight performance is a proper calibration of all the dynamic parameters of the system. In this section we describe how one can easily obtain an accurate estimation of: *i*) the accelerometer bias, *ii*) the gyroscope bias, *iii*) the motor/propeller characteristics *iv*) the mass and inertia matrix of the robot. For the sake of simplicity we will assume that the world Z-axis is aligned with the direction of gravity.

5.1 Gyroscope bias calibration

The bias of the gyroscope is defined as the output of the sensor when not experiencing any rotation. This bias is influenced by many variable factors like, e.g. the temperature, and it is then affected by considerable drift. As a consequence the calibration of this sensor must be repeated often enough. We therefore decided to completely automate this operation: each time the *MkInterface node* is restarted, a fixed number N of gyro samples are registered. The mean of the readings is then taken as a bias, provided that their standard deviation is below a certain threshold. Otherwise an error is thrown.

5.2 Accelerometer bias calibration

As well-known, an accelerometer measures its acceleration w.r.t. the inertial frame apart from any gravity component. Any accelerometer is affected by a bias, defined as the output of the sensor during free-fall under gravity. In addition, a scaling factor and a roto-translation w.r.t. the UAV body frame can also be present after being mounted. Differently from the gyroscope case, the accelerometer bias is almost constant over time, and it is thus not necessary to repeat the calibration unless the mounting of the sensor is changed.



(a) (b)

Fig. 2. Motor calibration: input/output signals (a) and fitting model (b).

To describe the employed calibration procedure, let $\ddot{\boldsymbol{r}}_{\mathcal{B}} = {}^{\mathcal{W}}\boldsymbol{R}_{\mathcal{B}}\mathbf{T}\left(\mathbf{a}_m - \mathbf{b}\right) + \mathbf{g}$ where $\mathbf{b}$ is a bias vector, matrix $\mathbf{T}$ takes into account any rotation and scaling factor, and $\mathbf{g}$ is the nominal value of the gravity acceleration. When the robot is at rest ($\ddot{\boldsymbol{r}}_{\mathcal{B}} = \mathbf{0}$), it is $\mathbf{T}\mathbf{a}_m - \mathbf{T}\mathbf{b} = -{}^{\mathcal{W}}\boldsymbol{R}_{\mathcal{B}}^T\mathbf{g}$. Assuming that a measure of ${}^{\mathcal{W}}\boldsymbol{R}_{\mathcal{B}}$ is available, this equation is linear in the unknowns $\mathbf{T}$ and $\mathbf{Tb}$. One can then repeat N measurements of $\mathbf{a}_m$ for different (constant) robot orientations and stack them in matrix form:

$$\begin{pmatrix} \mathbf{a}_{m,1}^T & -1 \\ \vdots & \vdots \\ \mathbf{a}_{m,N}^T & -1 \end{pmatrix} \begin{pmatrix} \mathbf{T}^T \\ \mathbf{b}^T\mathbf{T}^T \end{pmatrix} = \begin{pmatrix} -\mathbf{g}^T\,{}^{\mathcal{W}}\boldsymbol{R}_{\mathcal{B},1} \\ \vdots \\ -\mathbf{g}^T\,{}^{\mathcal{W}}\boldsymbol{R}_{\mathcal{B},N} \end{pmatrix}.$$

If the left matrix has rank 4, a simple pseudoinversion solves the system in a least-square sense. We note that this requires at least four available measurements for different orientations of the accelerometer.

As a final remark, we note that a similar rotation and scaling factor are also in general present for the gyroscope. Nevertheless the estimation of these quantities is more difficult because of the absence of a reference ground truth, as it is gravity for the accelerometer case. On the other hand, differently from the case of the accelerometer, any remaining miscalibration in the gyroscope sensor was experimentally found to be negligible.

5.3 Motor calibration

We identified the static characteristics of the brushless controller/motor/propeller system by using a Nano17 force/torque sensor [2]. The identification of the static characteristic was obtained by feeding the brush-less controller with the reference signal represented (in green) in Fig. 2a. We recall that this reference is linearly proportional to the desired rotational speed of the propeller. For each step, the motor reference was kept constant for 5 s, allowing for an attenuation of the noise by averaging the corresponding sensor measurements. Between each phase, a resting phase of 10 s was introduced with a low rotational speed for the propeller. This was necessary in order to avoid a propeller overheating which would have distorted the results. The reference was designed in order to vary from 30 to 190 in steps of 10.

As expected, the relationship between the rotational speed and the generated force/torque can be well approximated by a quadratic function. Since we are interested in the inverse relation, necessary to generate the reference for the brush-less controllers, we report the fitting curve in Fig. 2b with the force as the independent variable. We decided to exclude from the calibration any reference signal smaller than 30 or greater that 180 (resulting in a force of 0.3131 and 6.03 N respectively). This also allows to obtain a better fitting in the range most likely encountered during normal flight. We also note that, with the chosen limitations on the maximum rotor speeds, and considering the weight of the robot, the motors approximately operate in the center of their dynamical range during hovering.

The ratio between generated forces and torques is also quite constant in the range of interest, as well-known in the literature, and results about 0.0174 m.

Finally, the dynamics between commanded and actual propeller speeds was found to be well approximated by a first order linear system with a time constant of 0.047 s. Even if present, we decided to neglect this effect in the control design.

5.4 Estimation of the mass and inertia tensor

In order to estimate the robot mass, we made use of an algorithm taken from classical adaptive control techniques [15]. We start by considering the *scalar* vertical dynamics of a UAV, obtained by projecting (2) along $\boldsymbol{z}_{\mathcal{W}}$. This can be rearranged as:

$$\ddot{z} - g = \frac{1}{m}u_1\boldsymbol{z}_{\mathcal{B}}^T\boldsymbol{z}_{\mathcal{W}} = \frac{1}{m}\upsilon = m_{inv}\upsilon. \tag{6}$$

For convenience, we will estimate the inverse of the mass $m_{inv} = 1/m$. Equation (6) depends on the vertical acceleration $\ddot{z}$ which, in practice, is usually a very noisy quantity. To obtain a dependence in terms of the linear vertical velocity $\dot{z}$, we multiply both sides of (6) by a low-pass filter $P(s) = a/(s+a)$

$$P(s)(\ddot{z} - g) = m_{inv}P(s)\upsilon. \tag{7}$$

The term $P(s)\ddot{z}$ in (7) can be expanded as

$$\frac{a}{s+a}\ddot{z} = \frac{as}{s+a}\dot{z} = a\left(\dot{z} - \frac{a}{s+a}\dot{z}\right) = a\left(\dot{z} - P(s)\dot{z}\right), \tag{8}$$

i.e., as a function of the sole velocity $\dot{z}$. Now, let $g_{fil} = P(s)g$, $\upsilon_{fil} = P(s)\upsilon$, and $\dot{z}_{fil} = P(s)\dot{z}$. Equation (7) can be rewritten as

$$a(\dot{z} - \dot{z}_{fil}) - g_{fil} = m_{inv}\upsilon_{fil}. \tag{9}$$

Assume an estimation of the (inverse of the) mass $\hat{m}_{inv}$ is available, and let the lhs of (9) be $y = a(\dot{z} - \dot{z}_{fil}) - g_{fil}$ and the rhs of (9) evaluated on the estimated mass be $\hat{y} = \hat{m}_{inv}\upsilon_{fil}$. Defining the error function

$$V(e) = \frac{1}{2}k_{est}\left(y - \hat{y}\right)^2, \quad k_{est,m} > 0,$$

one can minimize it in terms of m_{inv} by implementing the following gradient update rule

$$\dot{\hat{m}}_{inv} = -\frac{\partial V(e)}{\partial \hat{m}_{inv}} = k_{est,m}e\upsilon_{fil}. \tag{10}$$

Note that the implementation of (10) requires knowledge of *i*) the vertical (world frame) velocity $\dot{z}$, *ii*) the direction of the axis $\boldsymbol{z}_{\mathcal{B}}$, *iii*) the commanded thrust u_1, and *iv*) the evaluation of the low-pass filtered signals υ_{fil}, g_{fil} and $\dot{z}_{fil}$. In our implementation, the following values were used: $a = 0.06\pi$ and $k_{est,m} = 1$.

Finally, we note that the same algorithm can be used for obtaining an estimation of the robot inertia tensor. Indeed, because of the cylindrical symmetry of the robot,

[2] http://www.ati-ia.com

the inertia matrix results practically diagonal and with the same X and Y components. Under this assumption the gyroscopic term in (3) becomes negligible for a small Z component of ${}^{\mathcal{B}}\boldsymbol{\omega}_{\mathcal{BW}}$. The rotational dynamics can then be split among three independent scalar systems similar to (6), with the additional simplification of the absence of any gravity term.

6. STATE ESTIMATION

The control algorithm described in Sec. 4 requires knowledge, at high rate, of the full state of the robot, i.e., of the position $\boldsymbol{r}_{\mathcal{B}}$, the velocity $\dot{\boldsymbol{r}}_{\mathcal{B}}$, the orientation ${}^{\mathcal{W}}\boldsymbol{R}_{\mathcal{B}}$ and the angular velocity ${}^{\mathcal{B}}\boldsymbol{\omega}_{\mathcal{BW}}$. In this section we describe how to obtain an estimation of these quantities using measurements of linear acceleration and angular velocity available from the onboard IMU, and a low rate measurements of the position and orientation provided by any possible source (e.g., an external or onboard sensor).

6.1 Complementary filter in SE(3)

If some (possibly noisy) measurement of the robot orientation is available, the translational part of the robot dynamics $\boldsymbol{\chi}_T = (\boldsymbol{r}_{\mathcal{B}}, \dot{\boldsymbol{r}}_{\mathcal{B}})^T$ has the linear expression

$$\dot{\boldsymbol{\chi}}_T = \frac{d}{dt}\begin{pmatrix}\boldsymbol{r}_{\mathcal{B}}\\ \dot{\boldsymbol{r}}_{\mathcal{B}}\end{pmatrix} = \begin{pmatrix}\mathbf{0}_{3\times3} & \mathbf{I}_{3\times3}\\ \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3}\end{pmatrix}\boldsymbol{\chi}_T + \begin{pmatrix}\mathbf{0}_{3\times1}\\ {}^{\mathcal{W}}\boldsymbol{R}_{\mathcal{B}}\mathbf{T}\left(\mathbf{a}_m - \mathbf{b}\right) + \mathbf{g}\end{pmatrix} = \mathbf{A}\boldsymbol{\chi}_T + \boldsymbol{\upsilon}.$$

By assuming a low-rate and possibly noisy measurement of the position $\boldsymbol{r}_{\mathcal{B}}$ is available, it is straightforward to check that the system is observable and a simple linear state observer can be easily designed as:

$$\dot{\hat{\boldsymbol{\chi}}}_T = \mathbf{A}\hat{\boldsymbol{\chi}}_T + \boldsymbol{\upsilon} + \mathbf{K}_{est,p}\left(\boldsymbol{r}_{\mathcal{B}} - \hat{\mathbf{r}}_{\mathcal{B}}\right). \tag{11}$$

We note that one can also estimate the accelerometer bias $\mathbf{b}$. Indeed by considering the extended state $\boldsymbol{\chi}'_T = (\boldsymbol{r}_{\mathcal{B}}, \dot{\boldsymbol{r}}_{\mathcal{B}}, \mathbf{b})^T$, one has

$$\dot{\boldsymbol{\chi}}'_T = \frac{d}{dt}\begin{pmatrix}\boldsymbol{r}_{\mathcal{B}}\\ \dot{\boldsymbol{r}}_{\mathcal{B}}\\ \mathbf{b}\end{pmatrix} = \begin{pmatrix}\mathbf{0}_{3\times3} & \mathbf{I}_{3\times3} & \mathbf{0}_{3\times3}\\ \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & -{}^{\mathcal{W}}\boldsymbol{R}_{\mathcal{B}}\mathbf{T}\\ \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3}\end{pmatrix}\boldsymbol{\chi}'_T + \begin{pmatrix}\mathbf{0}_{3\times1}\\ {}^{\mathcal{W}}\boldsymbol{R}_{\mathcal{B}}\mathbf{T}\mathbf{a}_m + \mathbf{g}\\ \mathbf{0}_{3\times1}\end{pmatrix} = \mathbf{A}'\boldsymbol{\chi}_T + \boldsymbol{\upsilon}'.$$

It is easy to prove observability w.r.t. the position $\boldsymbol{r}_{\mathcal{B}}$ measurement in this case, too, thus allowing to implement an observer similar to (11).

As for the orientation, we implemented the complementary filter in $SO(3)$ described in [10], in particular the so-called *passive complementary filter* without gyro bias estimation (already compensated for during the calibration of Sec. 5), and using the quaternion-based formulation. Let ${}^{\mathcal{W}}q_{\mathcal{B}} = ({}^{\mathcal{W}}q_{\mathcal{B},s}, {}^{\mathcal{W}}\mathbf{q}_{\mathcal{B},v})$ be the unit-norm quaternion corresponding to ${}^{\mathcal{W}}\boldsymbol{R}_{\mathcal{B}}$ with scalar and vector part ${}^{\mathcal{W}}q_{\mathcal{B},s}$ and ${}^{\mathcal{W}}\mathbf{q}_{\mathcal{B},v}$, respectively. As well-known, the kinematics of the quaternion is given by:

$${}^{\mathcal{W}}\dot{q}_{\mathcal{B}} = \frac{1}{2}\,{}^{\mathcal{W}}q_{\mathcal{B}} \otimes \left(0,\, {}^{\mathcal{B}}\boldsymbol{\omega}_{\mathcal{BW}}\right)$$

where $\otimes$ indicates the quaternion product defined as:

$$p \otimes q = \left(p_s q_s - \mathbf{p}_v^T\mathbf{q}_v,\, p_s\mathbf{q}_v + q_s\mathbf{p}_v + \mathbf{p}_v \times \mathbf{q}_v\right).$$

Assuming that a low-rate and possibly noisy measure of the orientation ${}^{\mathcal{W}}q_{\mathcal{B}}$ is available, we can define an orientation error as

$$\delta q = {}^{\mathcal{W}}\hat{q}^*_{\mathcal{B}} \otimes {}^{\mathcal{W}}q_{\mathcal{B}} = (\delta q_s, \boldsymbol{\delta}\mathbf{q})$$

where the conjugate is defined as $q^* = (q_s, -\mathbf{q}_v)$. The estimation of the orientation can then be updated as follows:

$${}^{\mathcal{W}}\dot{\hat{q}}_{\mathcal{B}} = \frac{1}{2}\,{}^{\mathcal{W}}\hat{q}_{\mathcal{B}} \otimes \left(0,\, {}^{\mathcal{B}}\boldsymbol{\omega}_{\mathcal{BW}} + 2\mathbf{K}_{est,q}\delta q_s\boldsymbol{\delta}\mathbf{q}_v\right). \tag{12}$$

In order to enforce the unit-norm constraint during the numerical integration, we slightly modified (12) by adding a penalty term proportional to the squared norm of the estimated quaternion:

$${}^{\mathcal{W}}\dot{\hat{q}}_{\mathcal{B}} = \frac{1}{2}\,{}^{\mathcal{W}}\hat{q}_{\mathcal{B}} \otimes \left(2\lambda(1 - \left\|\,{}^{\mathcal{W}}\hat{q}_{\mathcal{B}}\right\|^2), \boldsymbol{\omega}_m + 2\mathbf{K}_{est,q}\delta q_s\boldsymbol{\delta}\mathbf{q}_v\right)$$

where $\boldsymbol{\omega}_m$ is the angular velocity measured by the gyroscope corrected for the calibrated bias. Finally, we employed the following values: $\mathbf{K}_{est,p} = 126.3\mathbf{I}_3$, $\mathbf{K}_{est,v} = 3987.3\mathbf{I}_3$, and $\mathbf{K}_{est,q} = 62.8\mathbf{I}_3$.

The last part of the state to be estimated is the angular velocity ${}^{\mathcal{B}}\boldsymbol{\omega}_{\mathcal{BW}}$. In principle this quantity is already provided by the onboard gyroscope. In practice the output of this sensor during a typical flight is too noisy to be directly fed to the controller. Therefore, we low-pass filtered ${}^{\mathcal{B}}\boldsymbol{\omega}_{\mathcal{BW}}$ as $\dot{\hat{\boldsymbol{\omega}}} = 2\pi f_\omega(\boldsymbol{\omega}_m - \hat{\boldsymbol{\omega}})$ with $f_\omega = 150$ Hz.

6.2 Extension to bearing-only measurements

The state estimation algorithm described above requires a low-rate and possibly noisy direct measurement of the position and orientation of the robot. Since these measurements are not always available in a laboratory, we here detail a ready-to-use algorithm able to recover these quantities from a set of (at least 4) bearing measurements from some beacons with known identity and position in the world frame. This is, indeed, representative of many situations involving an onboard camera extracting features from the scene.

The reconstruction of the robot pose can be obtained by exploiting standard techniques from geometrical computer vision. The basic algorithm, described in [8], is here briefly summarized: first, consider a set of N points $\mathbf{p}$ belonging to the same 2D plane and assume that two bearing measurements are available for each point, namely $\mathbf{x}_{i,1}$ and $\mathbf{x}_{i,2}$ for $i = 1 \dots N$. These two bearings are associated to two frames (in our case the world and the body frame respectively). The plane is described by the following equation in the first reference frame:

$$\mathbb{P} = \left\{X_1 : \mathbf{n}_1^T X_1 = d\right\} \Leftrightarrow \frac{1}{d}\mathbf{n}_1^T X_1 = 1 \tag{13}$$

where X_1 contains the coordinates of the point $\mathbf{p}$ in the first frame, $\mathbf{n}_1$ is the unit normal vector to the plane and d is the distance of the plane from the origin of the first frame. Note that, since we have assumed a known beacon position in the world frame, all these quantities are also available.

Letting $({}^{\mathcal{W}}\boldsymbol{R}_{\mathcal{B}}, \boldsymbol{r}_{\mathcal{B}})$ represent the rotation/translation among the two frames, and exploiting (13), one has

$$X_2 = \left({}^{\mathcal{W}}\boldsymbol{R}_{\mathcal{B}} + \frac{1}{d}\boldsymbol{r}_{\mathcal{B}}\mathbf{n}_1^T\right)X_1 = \mathbf{H}X_1$$

where $\mathbf{H}$ is the so-called *homography matrix*. The homography $\mathbf{H}$ can be directly retrieved from a set of at least four bearing pairs (in the reference and current frame) by means of the standard *four-point algorithm for planar scene* as described in [8]. Once recovered, $\mathbf{H}$ can then be decomposed back into the rotation ${}^{\mathcal{W}}\boldsymbol{R}_{\mathcal{B}}$ and the translation $\boldsymbol{r}_{\mathcal{B}}$: let $(\mathbf{n}'_1, \mathbf{n}''_1)$ be a pair of unit-norm vectors such that $(\mathbf{n}_1, \mathbf{n}'_1, \mathbf{n}''_1)$ forms a right hand frame. Note that $(\mathbf{n}'_1, \mathbf{n}''_1)$ are a basis of the null space of ${\mathbf{n}_1}^T$. By projecting the homography matrix along these vectors, we obtain

$$\mathbf{H}\,(\mathbf{n}'_1 \;\; \mathbf{n}''_1) = (\mathbf{m}'_1 \;\; \mathbf{m}''_1) = {}^{\mathcal{W}}\boldsymbol{R}_{\mathcal{B}}\,(\mathbf{n}'_1 \;\; \mathbf{n}''_1)\,.$$

By then taking $\mathbf{m}_1 = \mathbf{m}'_1 \times \mathbf{m}''_1$, ${}^{\mathcal{W}}\boldsymbol{R}_{\mathcal{B}}$ is finally recovered as

$${}^{\mathcal{W}}\boldsymbol{R}_{\mathcal{B}} = (\mathbf{m}_1 \;\; \mathbf{m}'_1 \;\; \mathbf{m}''_1)\,(\mathbf{n}_1 \;\; \mathbf{n}'_1 \;\; \mathbf{n}''_1)^T\,.$$

Once ${}^{\mathcal{W}}\boldsymbol{R}_{\mathcal{B}}$ has been found, $\mathbf{t}$ is simply given by $\mathbf{t} = d\,(\mathbf{H} - {}^{\mathcal{W}}\boldsymbol{R}_{\mathcal{B}})$ where d is assumed known, as explained above. This then allows to recover a measurement of the robot pose $({}^{\mathcal{W}}\boldsymbol{R}_{\mathcal{B}}, \boldsymbol{r}_{\mathcal{B}})$ that can be used in the complementary filter described in Sec. 6.1 .

7. EXPERIMENTAL CASES STUDIES

In this section we present some experimental results obtained with the proposed platform. In the first two experiments we used an external tracking system [3] to provide pose measurements directly to the estimator described in 6.1. For the last experiment, instead, we used the external tracking system *only* for simulating low-rate and noisy bearing measurements similar to those that would be provided by typical onboard localization systems (e.g., vision-based) available in the literature.

The aim of the first experiment was to prove the performance of the mass estimation algorithm in hovering conditions and during arbitrary motion. To this end, we intentionally initialized the system from an initial guess for the estimated mass of 1 kg, i.e. 0.308 kg less than the nominal value. The result of the experiment is shown in Fig. 3. At the beginning of the experiment, the controller commanded a vertical thrust not sufficient for the robot to take off due to the low value of the estimated mass. After the take-off, the robot was then commanded to hover at a constant position until the estimated mass had reached its final value (about 1.394 kg). The convergence of the estimation can be detected by setting a threshold on its variance. We then manually controlled the cartesian velocity of the robot in a random way for verifying that the mass estimation was remaining constant despite the erratic motion. Figure 3c shows the actual robot velocity as estimated via the techniques illustrated in Sec. 6.

In the second experiment we show the tracking performance of our control algorithm while following an 8-shaped horizontal trajectory in X and Y (see Fig. 4). The 8 shape has a bounding box of size 1 m×1 m and is completed in 8 s. Both the Z position and the yaw reference were varied during motion by following sinusoidal-like trajectories with an amplitude of 0.2 m and frequency of 1 rad, respectively. The timing law was chosen to ensure the needed smoothness for the reference sent to the controller, i.e., continuous up to the fourth order for the position and up to the second order for the yaw angle. In Fig. 5 some snapshots of this experiment are shown and correspond to the phases of

[3] `http://www.vicon.com/`

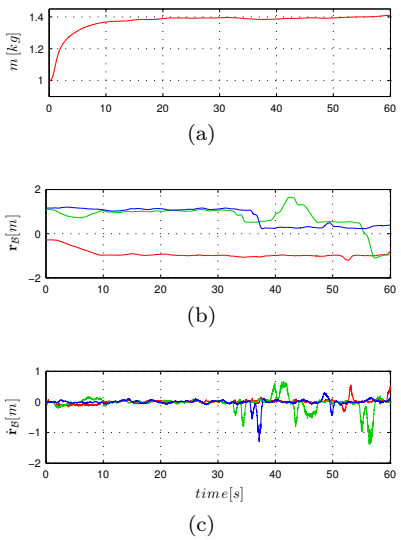


Fig. 3. Mass estimation during a normal flight. Fig. (a): estimated mass. Fig. (b): UAV position. Fig. (c) and UAV velocity.

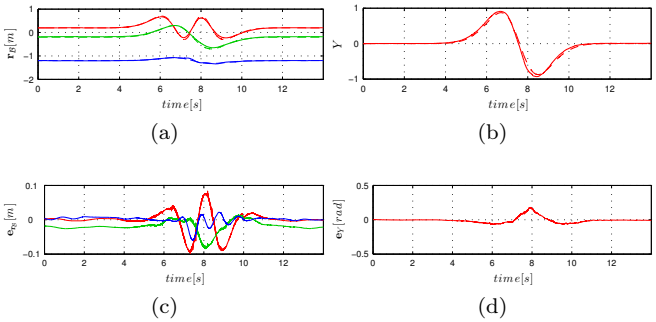


Fig. 4. Tracking of an eight shaped trajectory. Fig. (a): UAV cartesian position. Fig. (b): UAV yaw angle. Fig. (c): position error. Fig. (d): yaw angle error.

maximum tilt angles for the UAV, i.e. at the four sharp curves of the 8-shape.

As a final experiment, we report in Fig. 6 the results of the state estimation relying on bearing measurements. For this experiment the desired robot trajectory was generated manually with a joystick. We emulated the presence of 4 virtual beacons placed at 10 cm above ground and forming a square of 2 m size. The bearing measurement rate was set to 30 Hz with an additional gaussian angular noise with 1.5 deg STD. The pose of the robot was calculated from the simulated bearings by using the algorithm described in Sec. 6.2 and implemented in a Simulink model. From the comparison with the ground truth in the plots of Fig. 6, one can note that the pose estimation algorithm based on the Homography decomposition was still able to reconstruct a high-quality estimation of the robot state, and consequently to allow for a successful flight.

8. CONCLUSIONS

In this paper we have presented a completely open-source, ready-to-use and cheap platform for prototyping complex algorithms for single or multiple quadrotor UAVs, and relying on only onboard hardware. The

Fig. 5. Snapshots of the quadrotor executing an 8-shaped trajectory

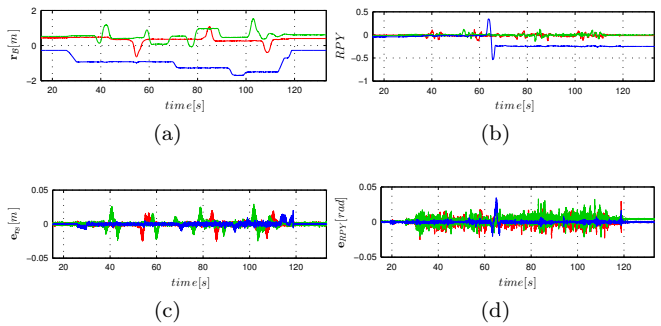


Fig. 6. State estimation using virtual bearings. Fig. (a): UAV cartesian position. Fig. (b): UAV orientation in roll-pitch-yaw representation. Fig. (c): position error. Fig. (d): orientation error.

platform can be fully customized from high-level behaviors to low-level motor control. The hardware and software architecture was described in detail with all the needed calibration and tuning procedures. Furthermore, we also freely distribute the whole software package for the interested reader at the SVN repository `https://svn.kyb.mpg.de/kyb-robotics/`. This allows an easy porting of our solution to any other lab interested in developing quadrotor applications for research purposes.

9. ACKNOWLEDGEMENTS

The authors want to thank Martin Riedel for his support with the TeleKyb framework.

REFERENCES

[asc] Ascending Technology. URL `http://www.asctec.de/home-en`.

[1] P.J. Bristeau, F. Callou, D. Vissière, and N. Petit. The Navigation and Control technology inside the AR.Drone micro UAV. In *18th IFAC World Congress*, Aug-Sep 2011.

[2] A. Franchi, C. Masone, V. Grabe, M. Ryll, H. H. Bülthoff, and P. Robuffo Giordano. Modeling and control of UAV bearing-formations with bilateral high-level steering. *The International Journal of Robotics Research, Special Issue on 3D Exploration, Mapping, and Surveillance*, 31(12):1504–1525, 2012.

[3] V. Grabe, H. H. Bülthoff, and P. Robuffo Giordano. On-board velocity estimation and closed-loop control of a quadrotor UAV based on optical flow. In *2012 IEEE Int. Conf. on Robotics and Automation*, pages 491–497, St. Paul, MN, May 2012.

[4] V. Grabe, M. Riedel, H. H. Bülthoff, P. Robuffo Giordano, and A. Franchi. The TeleKyb framework for a modular and extendible ROS-based quadrotor control. In *6th European Conference on Mobile Robots*, Barcelona, Spain, Sep. 2013.

[5] D. J. Lee, A. Franchi, H. I. Son, H. H. Bülthoff, and P. Robuffo Giordano. Semi-autonomous haptic teleoperation control architecture of multiple unmanned aerial vehicles. *IEEE/ASME Trans. on Mechatronics, Focused Section on Aerospace Mechatronics*, 18(4): 1334–1345, 2013.

[6] T. Lee, M. Leokyand, and N. H. McClamroch. Geometric tracking control of a quadrotor UAV on SE(3). In *49th IEEE Conf. on Decision and Control*, pages 5420–5425, Atlanta, GA, Dec. 2010.

[7] H. Lim, J. Park, D. Lee, and H. J. Kim. Build Your Own Quadrotor: Open-Source Projects on Unmanned Aerial Vehicles. *IEEE Robotics & Automation Magazine*, 19(3):33–45, 2012.

[8] Y. Ma, S. Soatto, J. Kosecka, and S. Sastry. *An invitation to 3D vision, from images to models*. Springer, New York, 2003. ISBN 0-387-00893-4.

[9] R. Mahony, T. Hamel, and J.-M. Pflimlin. Complementary filter design on the special orthogonal group SO(3). In *44th IEEE Conf. on Decision and Control*, pages 1477–1484, Seville, Spain, Dec. 2005.

[10] R. Mahony, T. Hamel, and J. Pflimlin. A study of non-linear complementary filter design for kinematic systems on the special orthogonal group. Technical report, Information Signaux et Systèmes de Sophia Antipoles, Université de Nice., Nov. 2006.

[11] R. Mahony, V. Kumar, and P. Corke. Multirotor Aerial Vehicles: Modeling, Estimation, and Control of Quadrotor. *IEEE Robotics & Automation Magazine*, 19(3):20–32, 2012.

[12] N. Michael, J. Fink, and V. Kumar. Cooperative manipulation and transportation with aerial robots. In *2009 Robotics: Science and Systems*, Seattle, WA, Jun. 2009.

[13] I. Sa and P. Corke. System identification, estimation and control for a cost effective open-source quadcopter. In *2012 IEEE Int. Conf. on Robotics and Automation*, pages 2035–2041, May 2012.

[14] M. Schwager, B. J. Julian, and D. Rus. Optimal coverage for multiple hovering robots with downward facing cameras. In *2009 IEEE Int. Conf. on Robotics and Automation*, pages 3515–3522, Kobe, Japan, May 2009.

[15] J. J. E. Slotine and W. Li. *Applied nonlinear control*. Prentice Hall, 1991. ISBN 9780130408907.

[16] Riccardo Spica. Planning and control for aerial grasping with a quadrotor UAV. Master Thesis, DIAG, Università di Roma La Sapienza, 2012.